

Notes on Power of functions









 medium.com/@manveetdn/notes-on-power-of-functions-b4f9180acc01

Representation power of functions

Disclaimer: This is notes on “Power of Functions” Lesson (PadhAI onefourthlabs course “A First Course on Deep Learning”)



We came across known currently are MP Neuron, Perceptron, Sigmoid Models from those models we can draw some conclusion like data, task, Model, Loss, Learning Evaluation(called the six jars namely).

						
MP neuron	$\{0, 1\}$	Binary Classification	$\hat{y} = 1$ if $\sum_{i=1}^n x_i \geq b$ $\hat{y} = 0$ otherwise	$Loss = \sum_{i=1}^n 1_{(y \neq \hat{y})}$		$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$
Perceptron	Real inputs	Binary Classification	$\hat{y} = 1$ if $\sum_{i=1}^n w_i x_i \geq b$ $\hat{y} = 0$ otherwise	$Loss = \sum_{i=1}^n (y - \hat{y})^2$	Perceptron Learning Algorithm	$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$
Sigmoid	Real inputs	Classification/Regression	$y = \frac{1}{1 + e^{(-w^T x + b)}}$	$Loss = \sum_{i=1}^n (y - \hat{y})^2$	Gradient Descent	$Accuracy/RMSE$ 

(Source: **onefourthlabs**)

The even with these models, we can't handle **non-linearly separable data** that's than main drawback we face while using this models.

The main quest here is to find a better function which can even handle non linearly separable data.

$$w_{t+1} = w_t - \eta \Delta w_t$$

$$b_{t+1} = b_t - \eta \Delta b_t$$

$$\Delta w_t = \frac{\partial L}{\partial w}$$

$$\Delta b_t = \frac{\partial L}{\partial b}$$

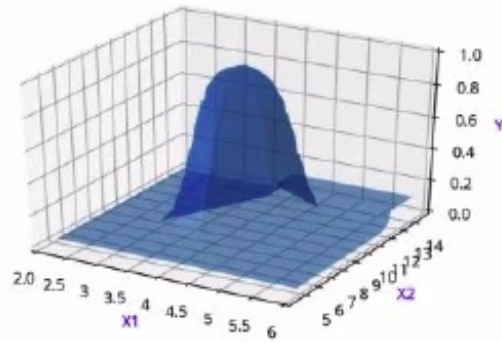
Gradient Descent (Source: **onefourthlabs**).

We will be using the gradient descent algorithm here to update the weights.

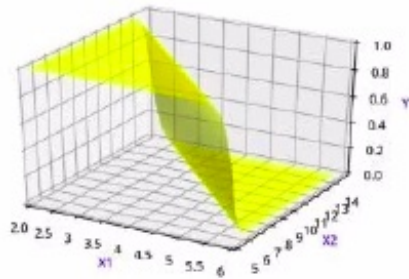
We will update the weight accordingly travelling in the opposite direction to the gradient.

That's how gradient descent works where partially **derivatives of loss function with w and b respectively** and subtracting those values from the previous values of w and b with a multiplying with small learning rate **eta** to generate new weights respectively $w(t+1)$ and $b(t+1)$.

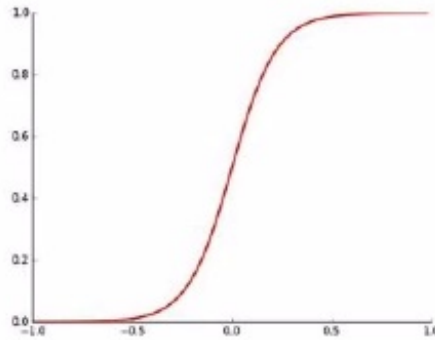
So in this case, we need to come up with continuous functions .Some of the continues functions are as shown below.



$$\hat{y} = sig_1(sig_2(x_1, x_2), sig_3(x_1, x_2), sig_4(x_1, x_2))$$



$$\hat{y} = \frac{1}{1+e^{-(-2*x_1+2*x_2+20)}}$$

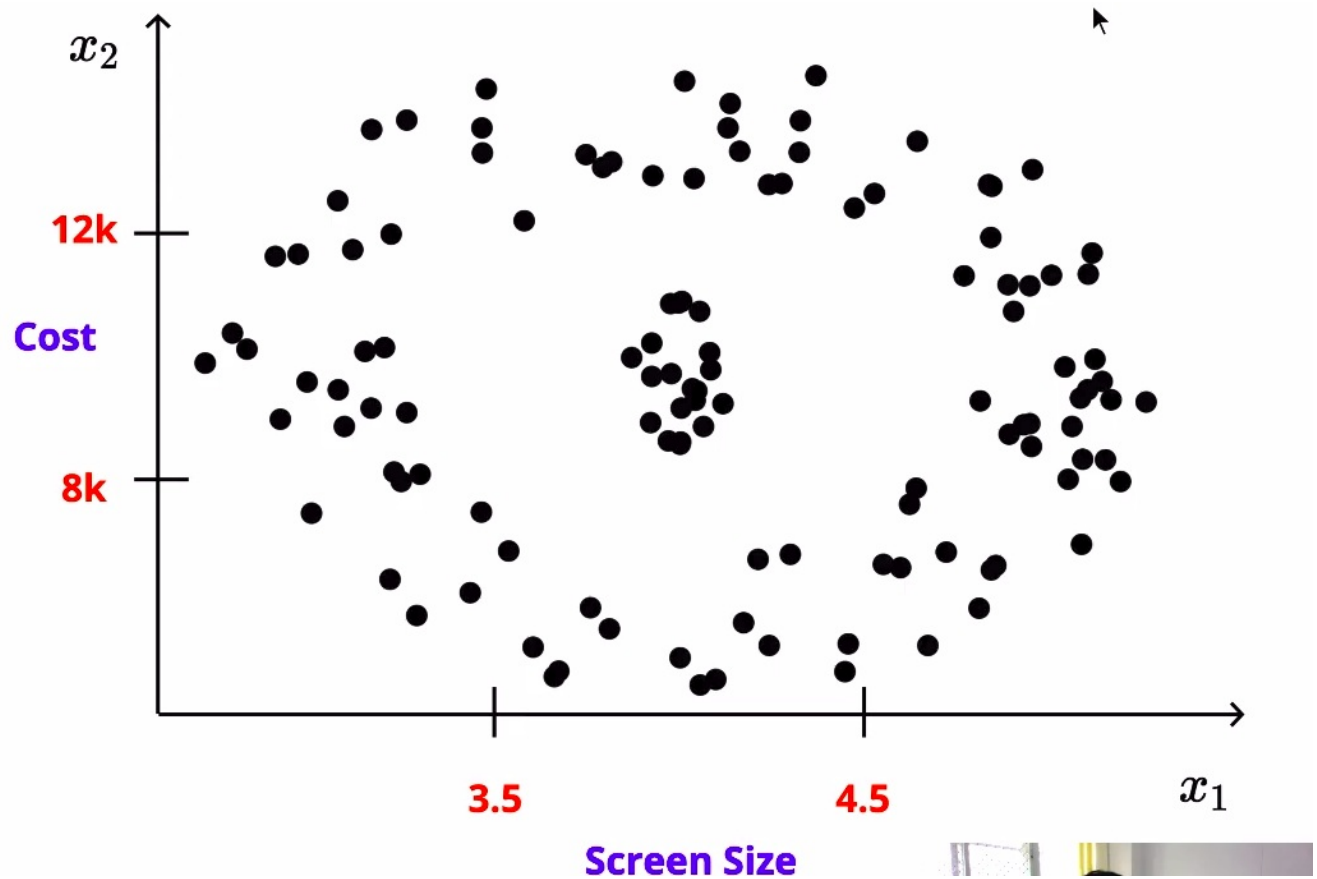


$$\hat{y} = \frac{1}{1+e^{-(2*x_1+5)}}$$

These three are some of examples of continuous functions (Source: **onefourthlabs**).

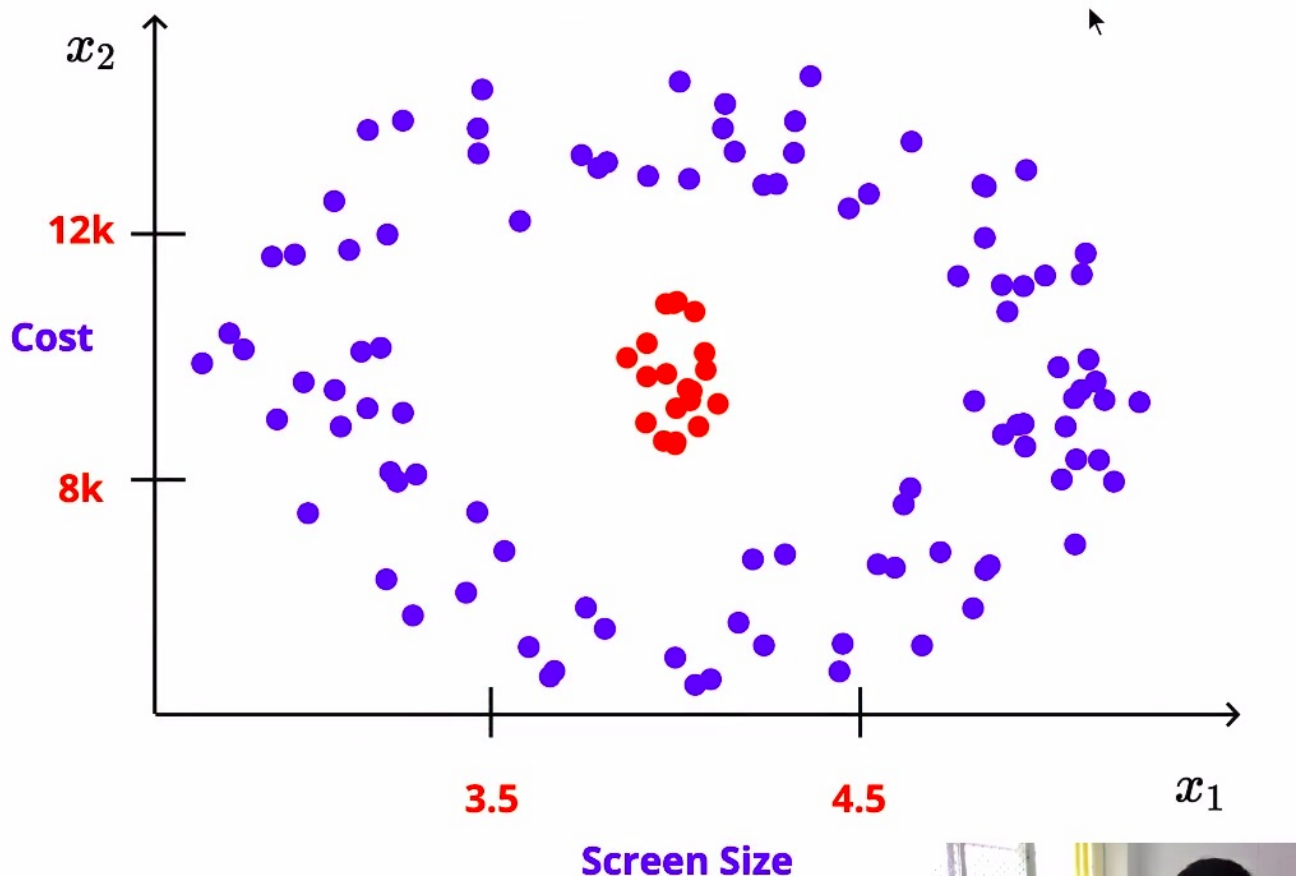
The first question we get is why do we need complex functions?

We need complex function for modelling complex relations. The simple example we can relate here is the mobile phones case, that is we can plot a graph between the **cost and the scree size** as show in the below graph.



Graphs between screen size and cost of mobile phones (Source: **onefourthlabs**)

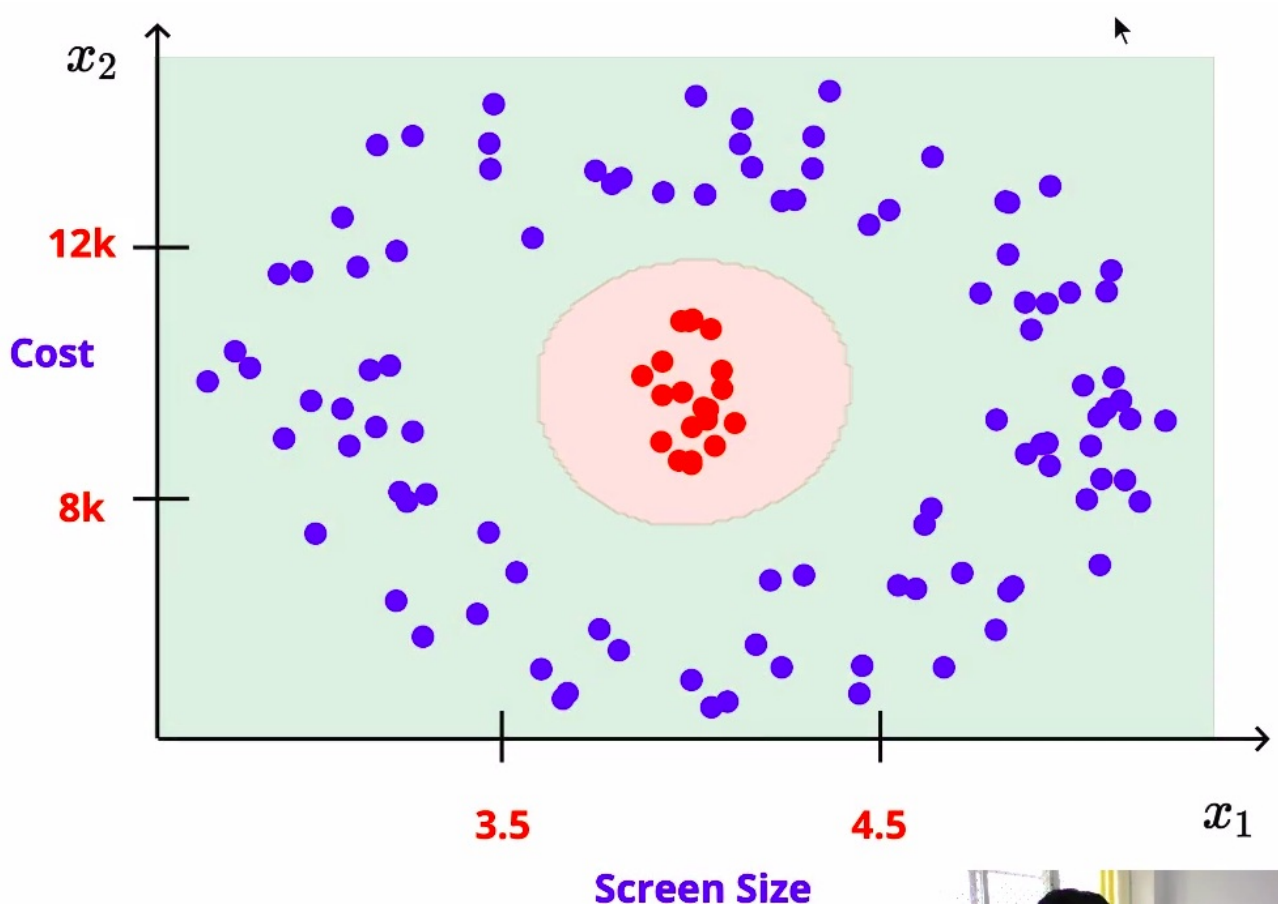
Here, as show in the graph the the screen sizes are **less than 3.5** , **between 3.5 and 4.5** and **greater than 4.5** case and the prices also ranging from **less that 8K** , **between 8K to 12K** and **more than 12K**. like thateach point plotted corresponds to the particular screen size and cost.

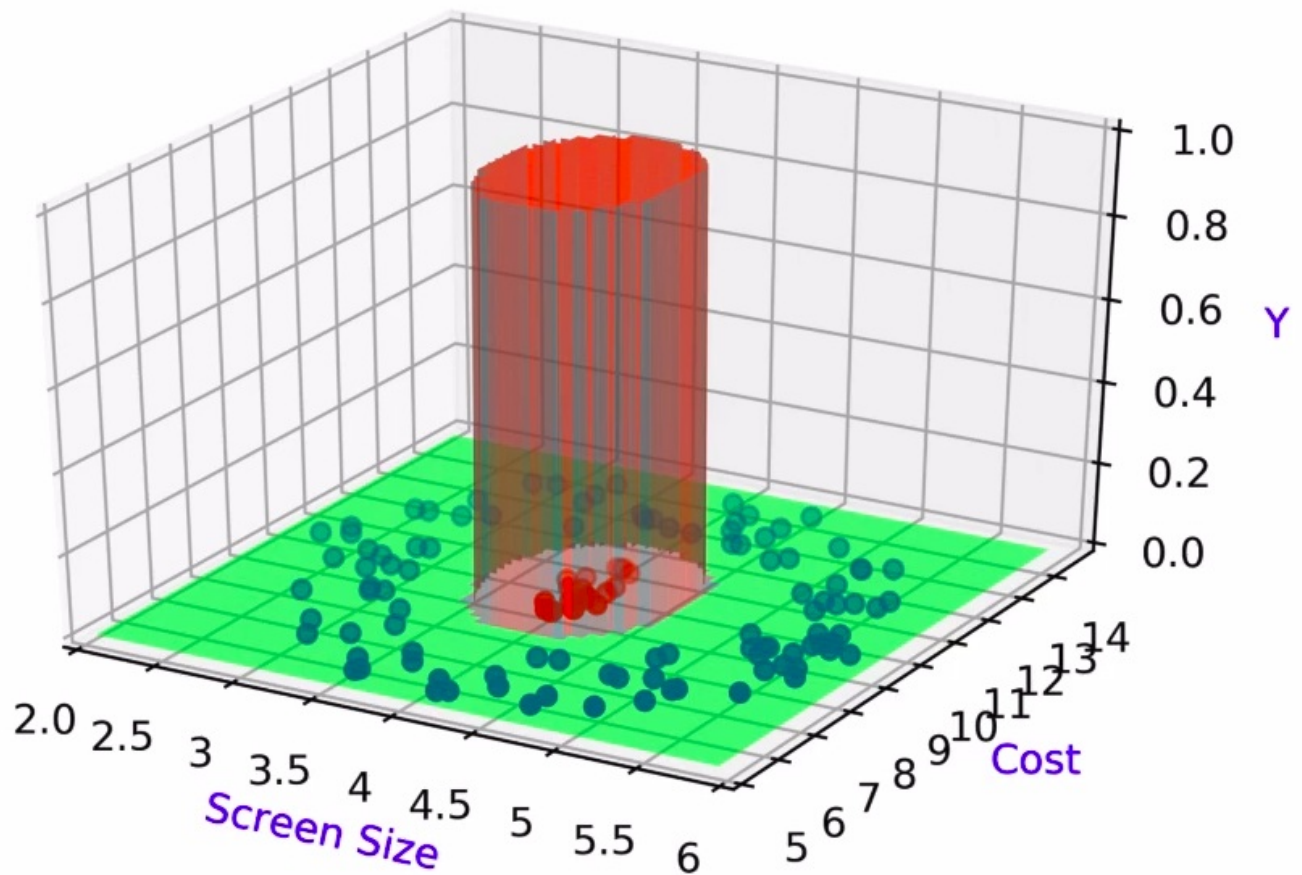


(Source : **onefourthlabs**)

If you are a person who likes only phone that ranges in price of 8K to 12K and the screen size between 2.5 and 4.5 (Red dots in the graph).

Here in this case there is no line you can draw to differentiate the red and blue points as shown because this data is non linearly separable data to separate the two points. so we need a function which separates data assigns **all blue points to a value of zero and all red points a value of one**. As shown in the below picture.



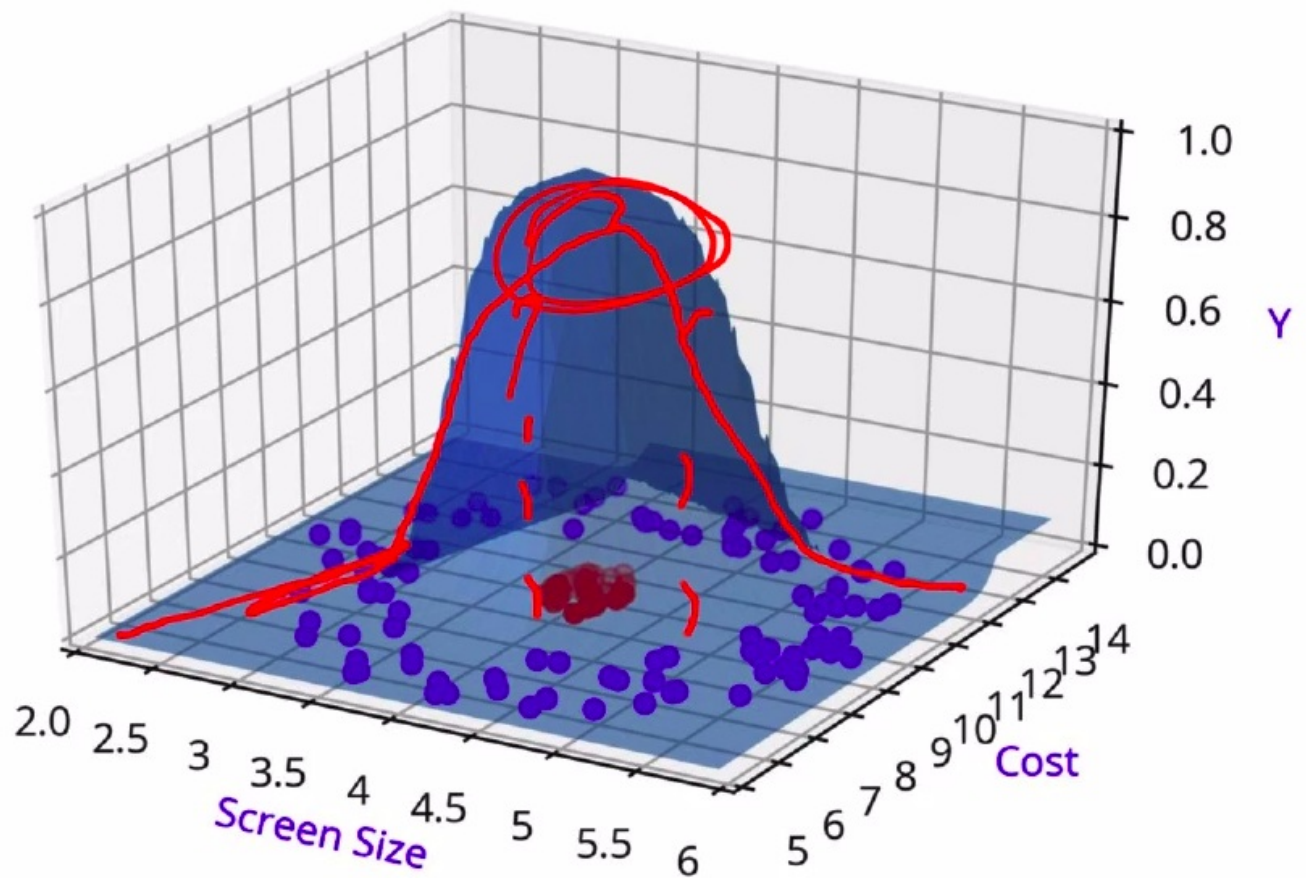
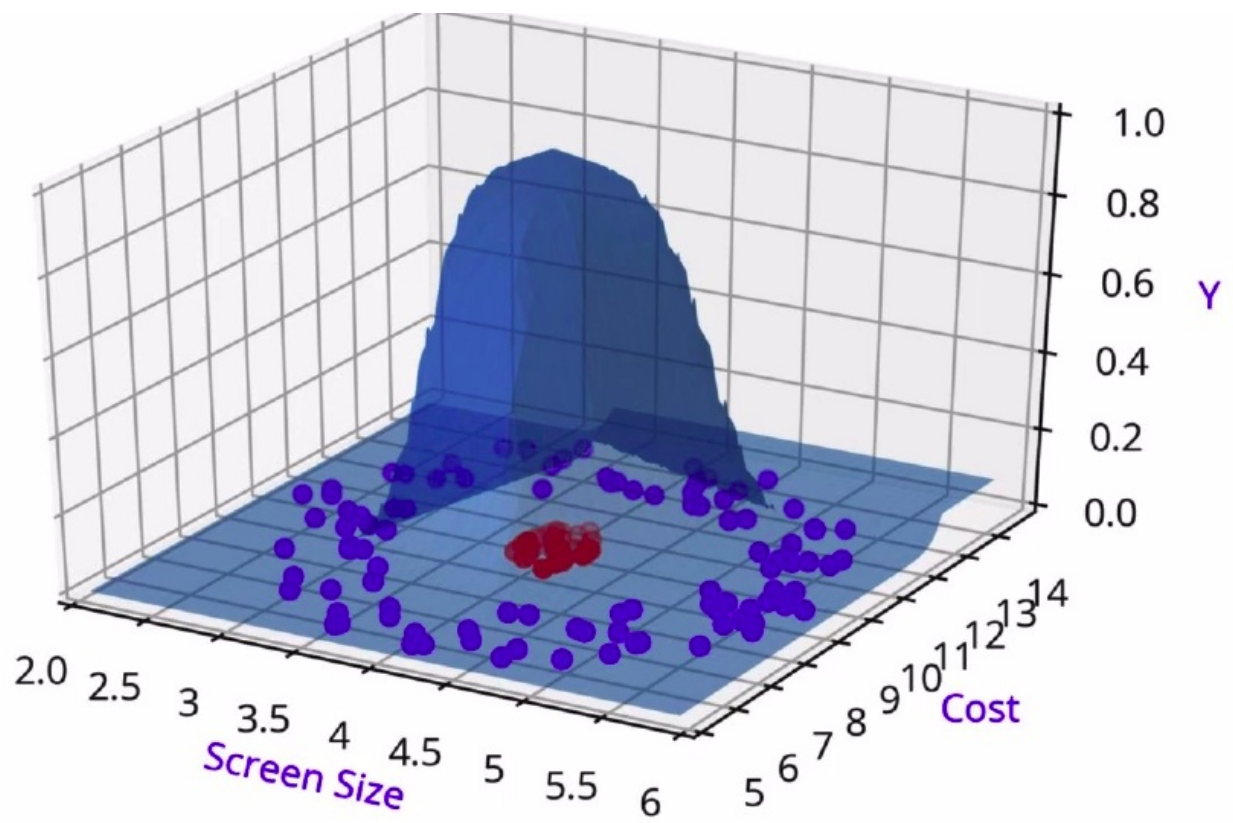


We need a classifier to separate the data like this (Source: **onefourthlabs**).

But actually, the right image at the top is not that convenient as it has sharp edges and its not a smooth function, which implies that **it is not differential at certain points**.

What we need is a smooth function which is differentiable at all the points

which is 0 in all the blue regions and gradually climbs up to 1 the the 1 should correspond to the red points in this case and everywhere else its 0.



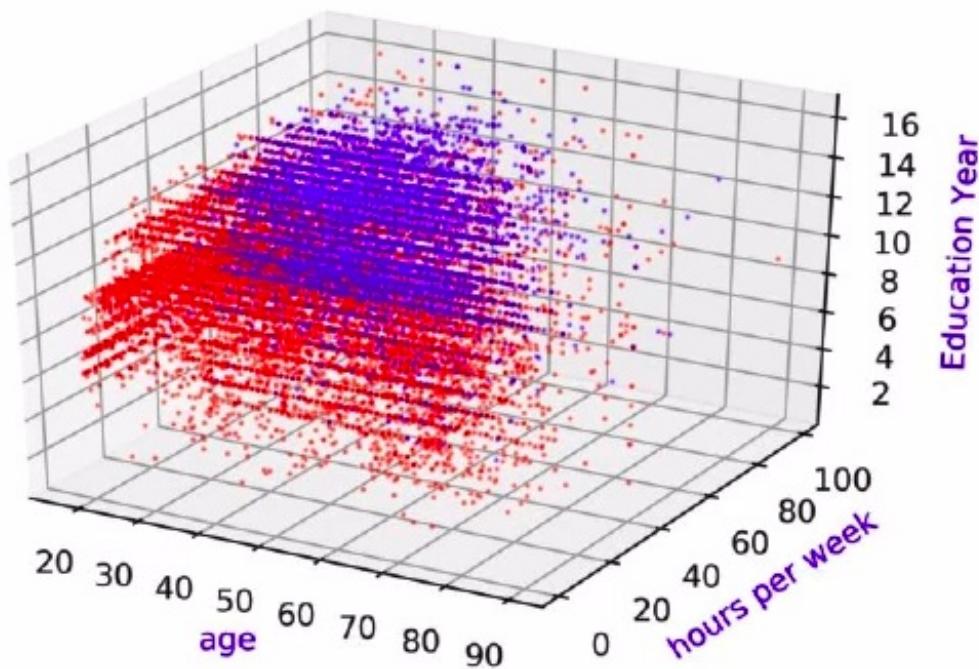
This is the smooth function we need to classify points (Source: **onefourthlabs**).

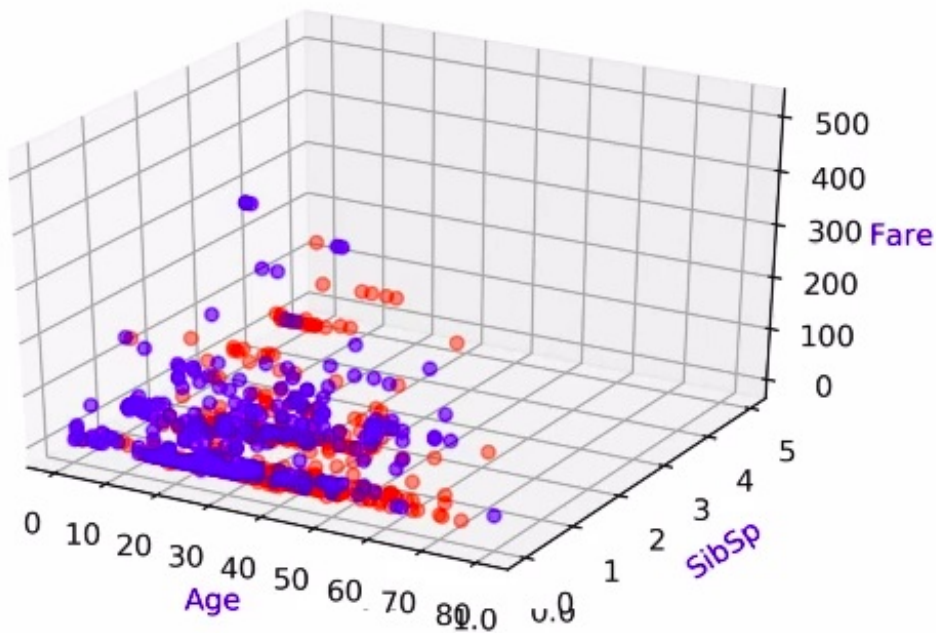
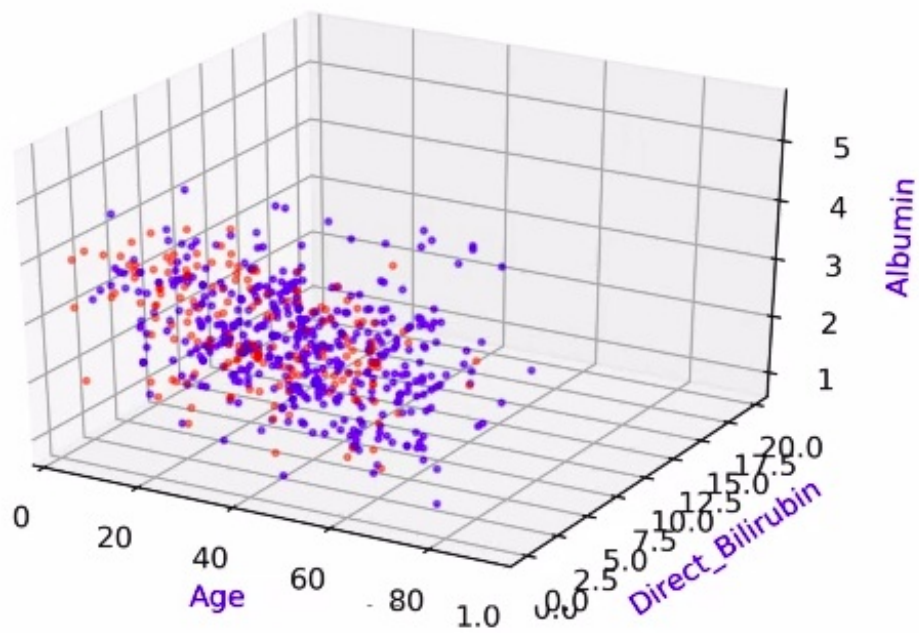
Like this the function corresponds to the red points and the blue points with zero and one, it is differentiable at all the points being a a smooth curve.

Even a sigmoid function fails in this case because the shape of sigmoid does not satisfy the the points to classify between red and blue in this case.

How do we even come up with complex functions??

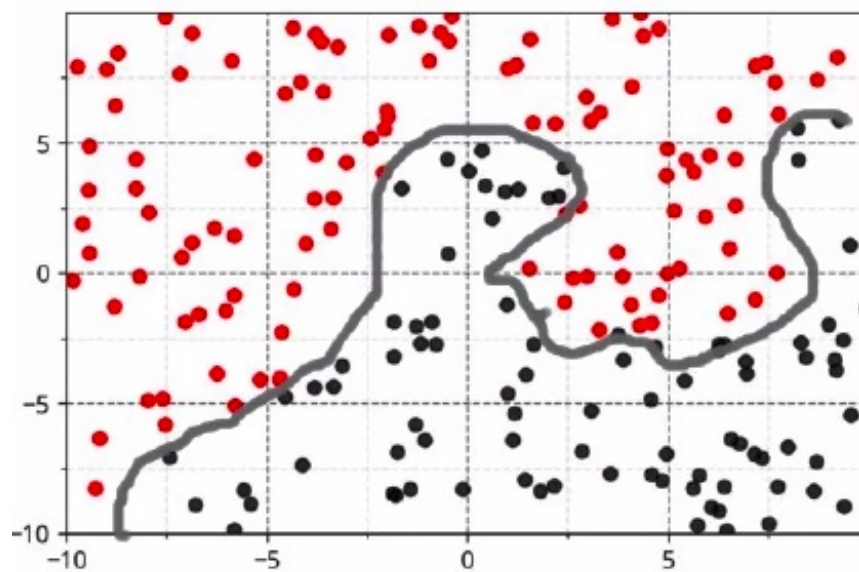
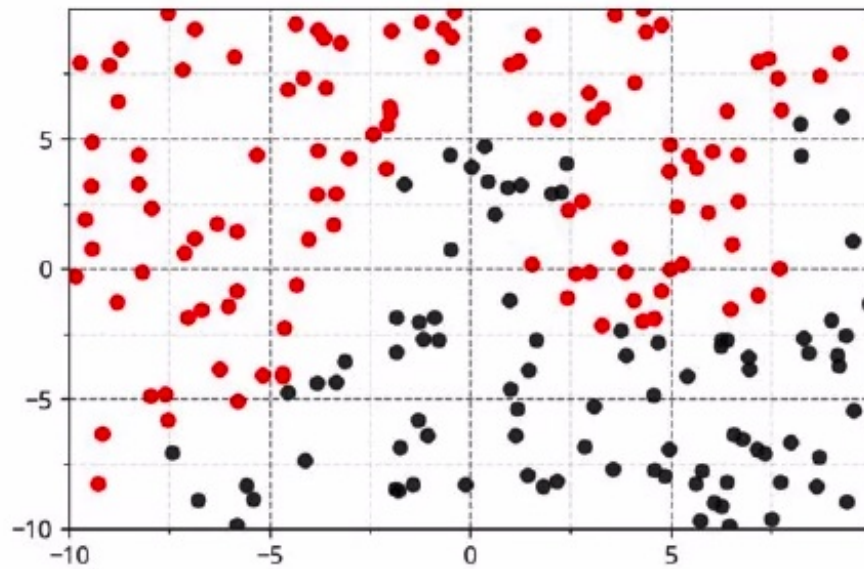
Many data sets from kaggle and real time examples ,problems . The datasets are plotted as below where sigmoid function also fails to classify them where we need a complex function to classify them.

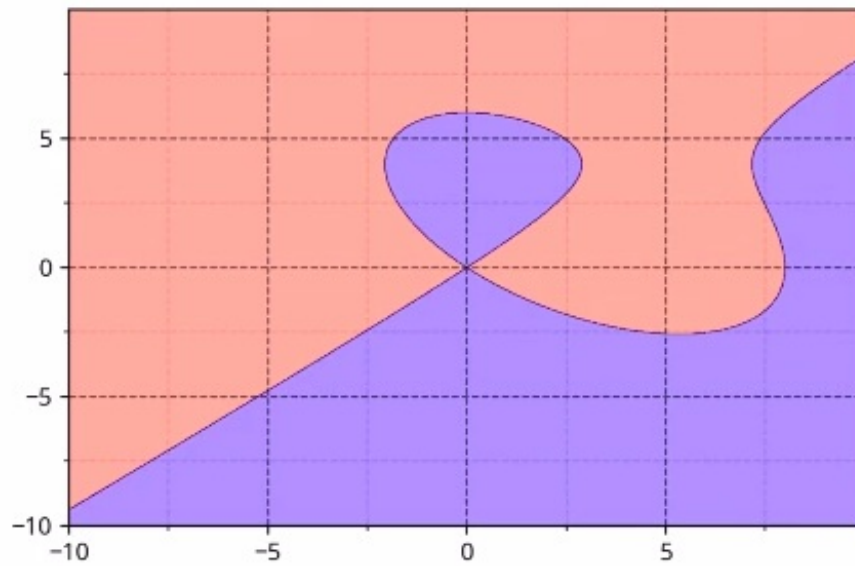




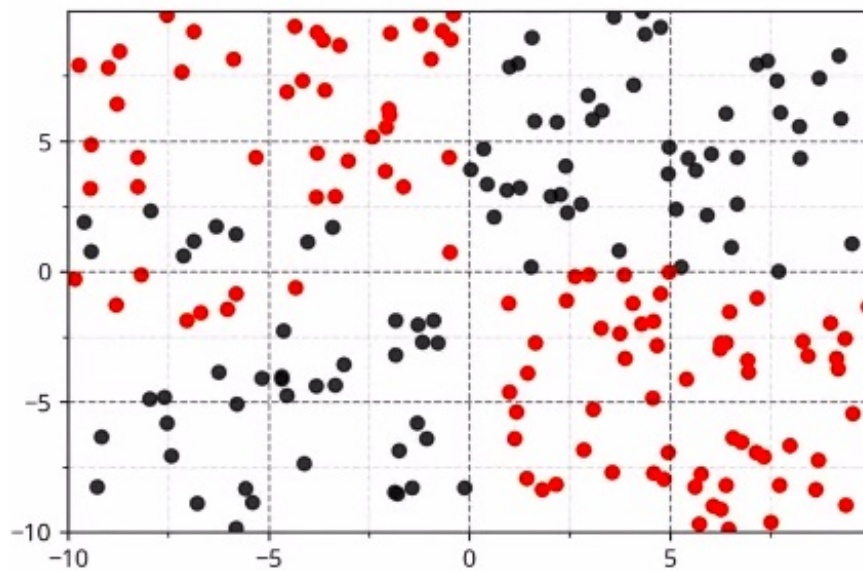
(i) first image is a dataset of calculating employee salary greater or less than 50K based on the data given based upon age, experience, company and various parameters. (ii) it is based on the Titanic dataset and should decide whether he should be diagnosed or not based on age and various parameters. (iii) Finally is the dataset of people's chance of living in a Titanic crash. (Source: **onefourthlabs**)

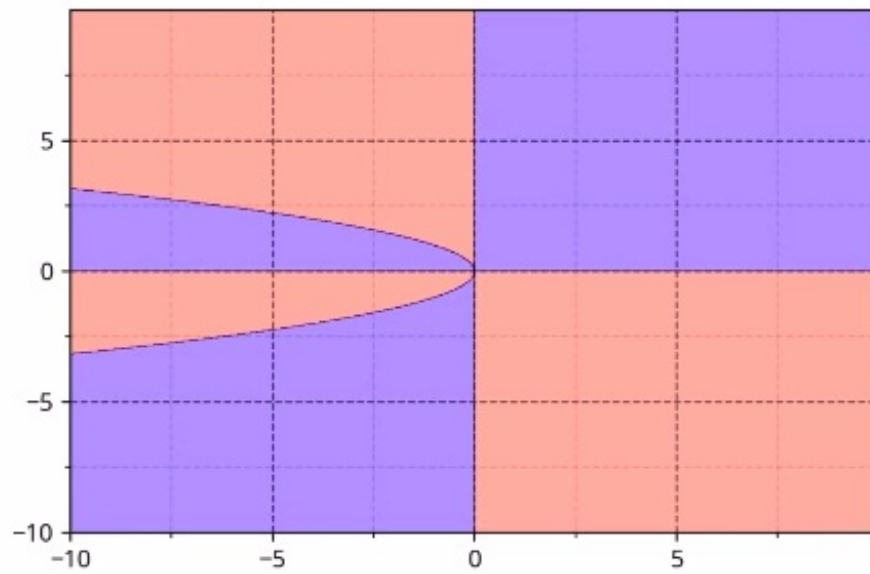
In the all above cases its very tough and impossible to classify data using MP neuron, Perceptron and Sigmoid functions.



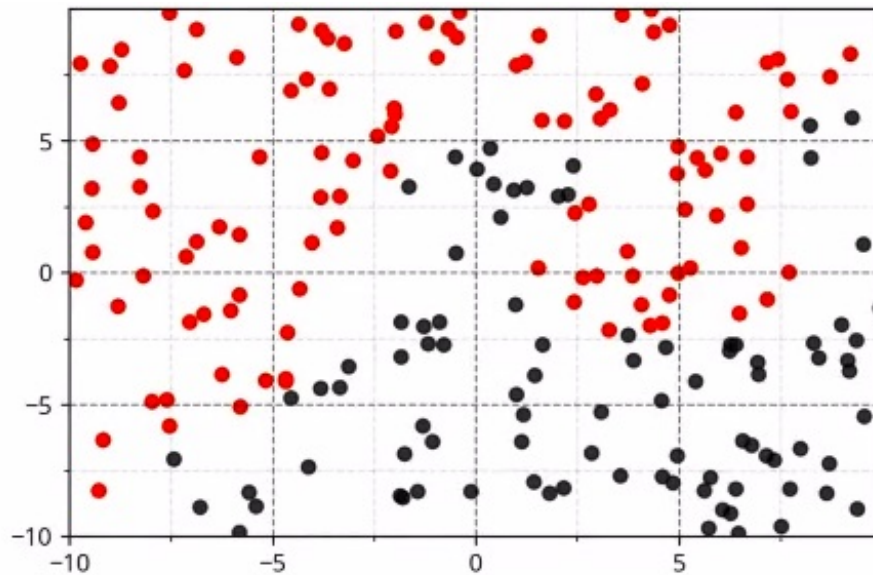


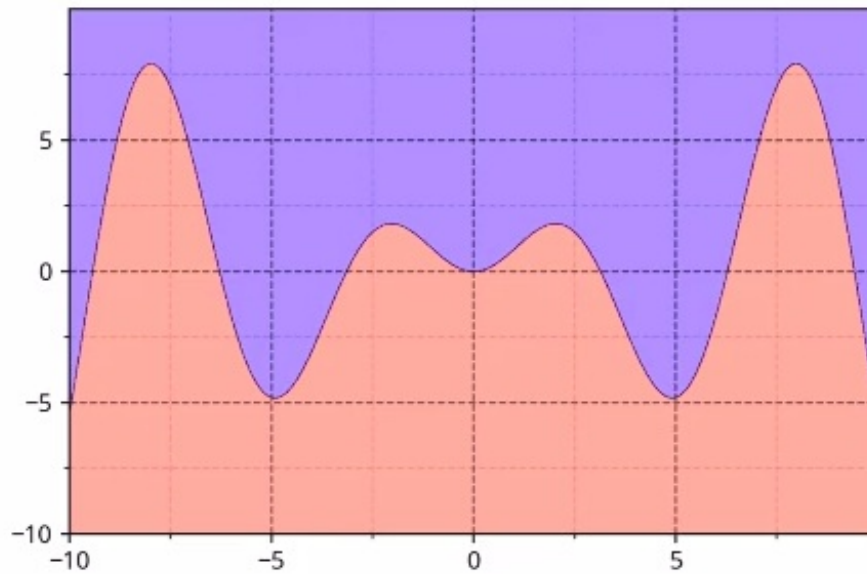
if we have a data set like this the we need to classify points lie 2nd image and then last image describes how the data is being classified (Source: **onefourthlabs**)





Similarly if we have the data as first image above then we need to classify as the above as shape the function classifies like that finally (Source: **onefourthlabs**)





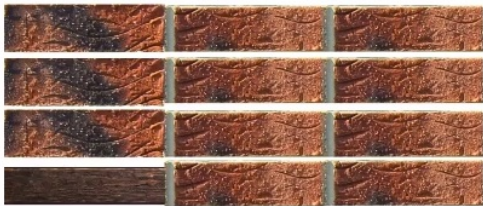
Similar to the above ones here also we need a function to classify like that. (Source: **onefourthlabs**)

The above are very complex function we can even define them and come to a conclusion that this is the equation and we don't even know where to start and those are the complex functions.

Constructing a Building:

A great building, a famous building, a tall building, a short building, a complex building, a simple building whatever may be the building is construction starts with a single brick, placing one brick on another and one beside another upon the other as per required shape and we will finish the construction how complex it may be.

Even complex building are built starting with single brick :P



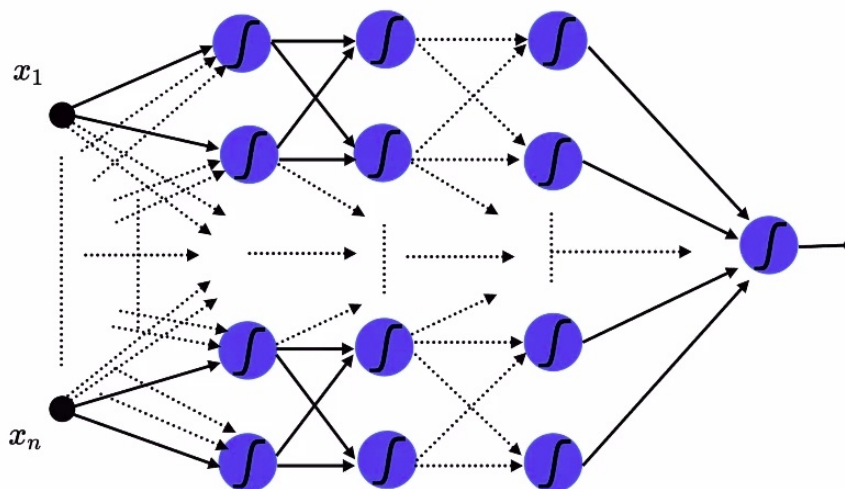
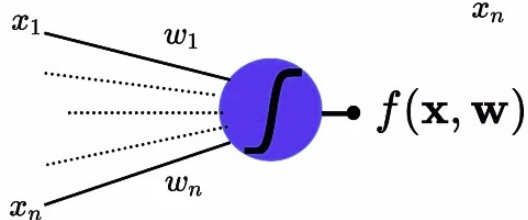
(c) One Fourth Labs

Source: OneFourthLabs (Source: **onefourthlabs**)

Same case with the complex function we need to built that with a starting at a small sigmoid function instead of bricks in this case. Like this we place sigmoid over sigmoid and sigmoid beside sigmoid and build a get network of sigmoid functions and there ends a great building called a **Neural Network**.

$$f(x_1, \dots, x_n) = \frac{1}{1 + e^{-(w_1 * x_1 + \dots + w_n * x_n + b)}}$$

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} * \mathbf{x} + b)}}$$

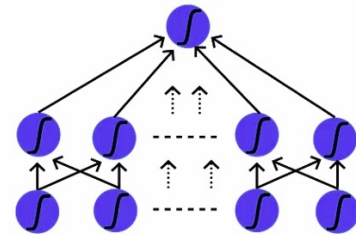
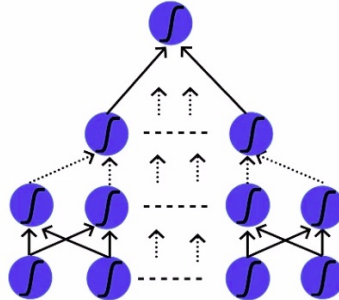
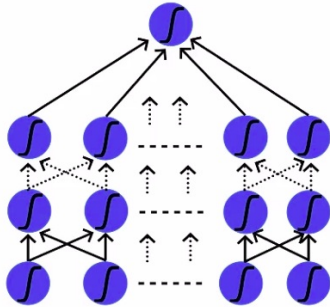
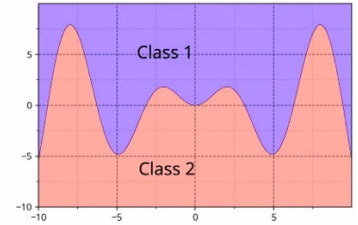
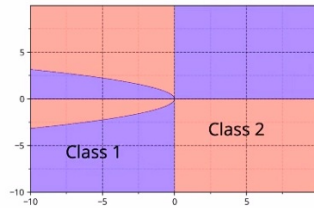
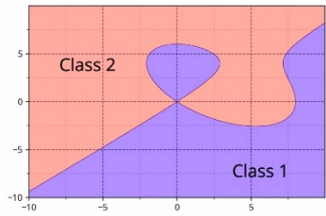


(Source: **onefourthlabs**)

Now our function is a deep neural network that takes x_1 to x_n as inputs and passes them through multiple transformations.

This is called **Universal Approximation Theorem**.

Like that, we will build different networks based on the classification requirement; all networks will not be the same. They vary in the number of layers of sigmoid neurons and the number of sigmoid neurons per layer. That's how we build a great and a deep neural network.



$$f(x_1, \dots, x_n) = \frac{1}{1 + e^{-(w_1 x_1 + \dots + w_n x_n + b)}} \equiv f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \equiv \sigma$$

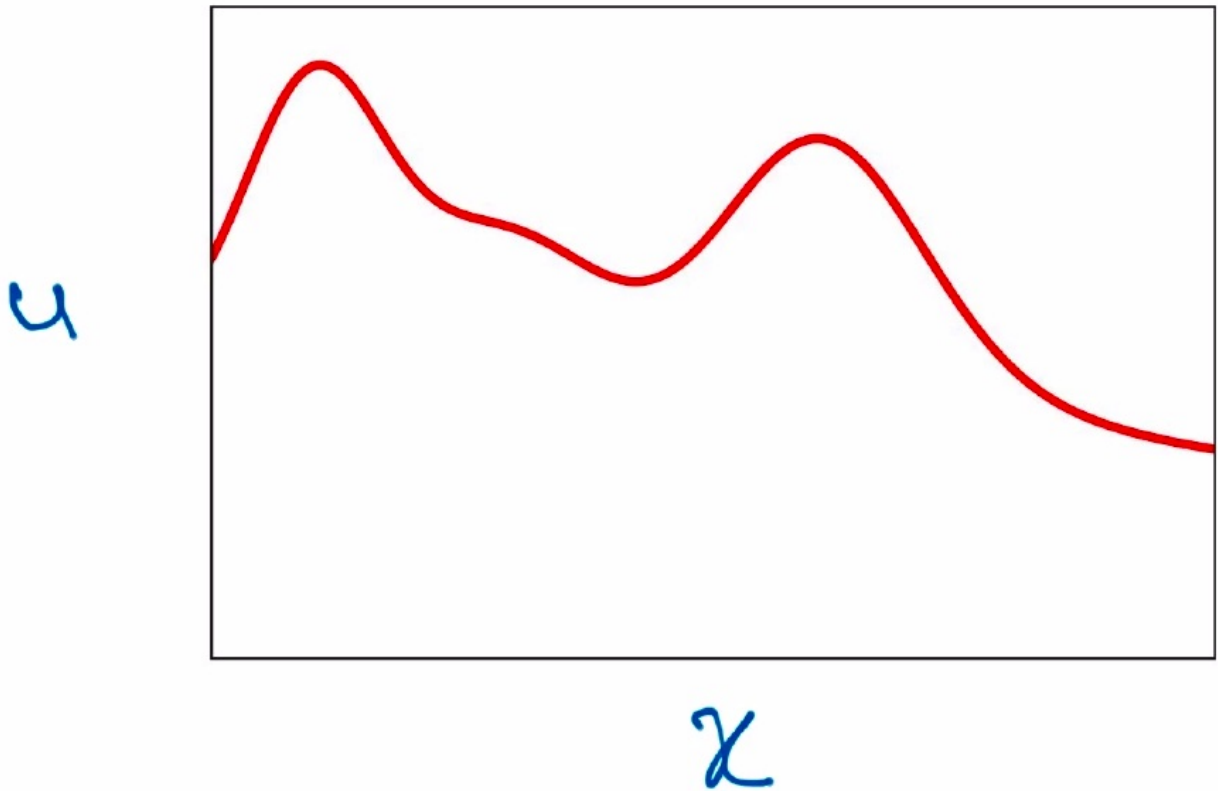


(Source: **onefourthlabs**)

Universal Approximation theorem:

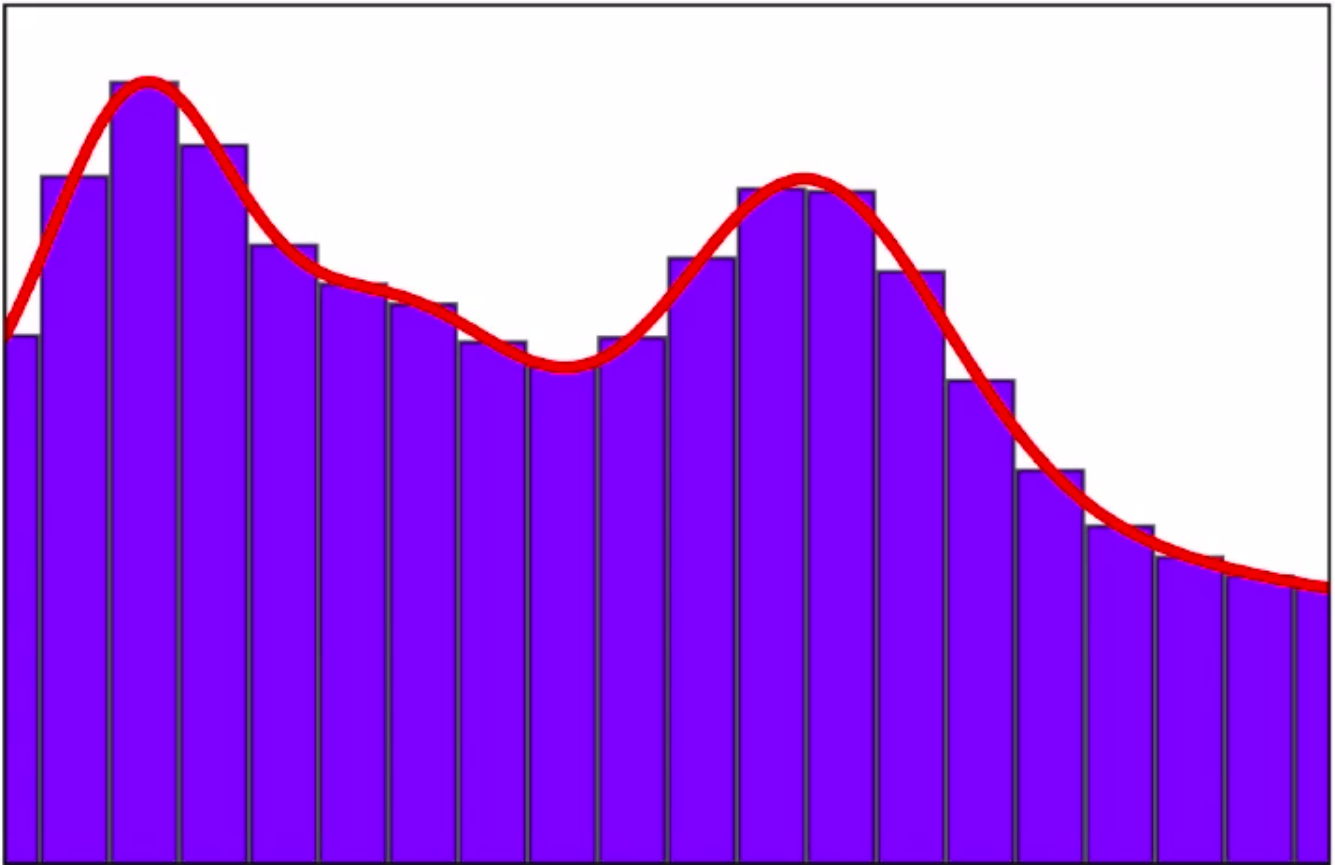
Here, Neurons act as building block many such building blocks become a function and make a function from inputs to outputs, the strategy is to try many such functions and see which one works the best. that's is many neurons contribute forming a neural network, to solve any complex function.

Informal Proof:



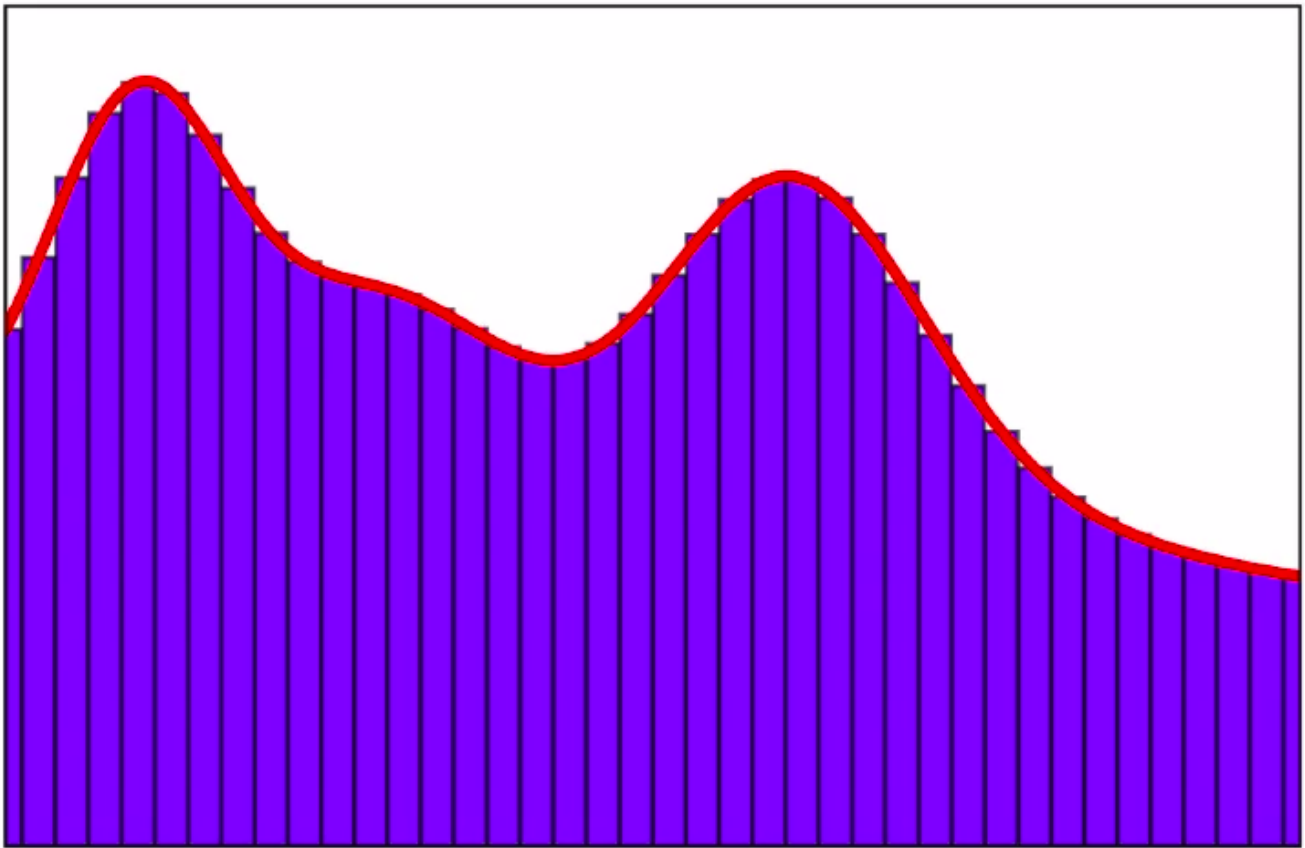
(Source: **onefourthlabs**)

what function we see here is true relationship between x and y . whatever the solution is the true relationship between x and y [$y = f(x)$]. we don't know what this function is and we we will approximate this function using different functions.



(Source: **onefourthlabs**)

Like this we will approximate the true function with different small functions such as small violet coloured bars.

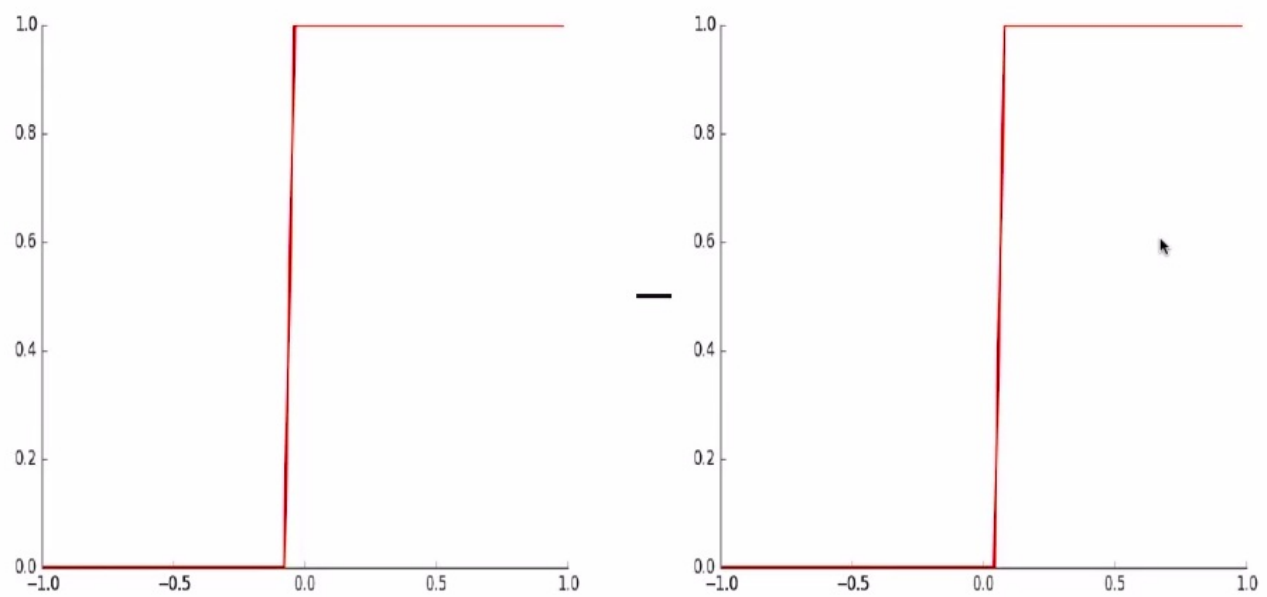


(Source: **onefourthlabs**)

The more number of function you take, the more accurate you will validate the function that's how you come out with the complex function. With more number of thin bars we will come too closer to the true approximation.

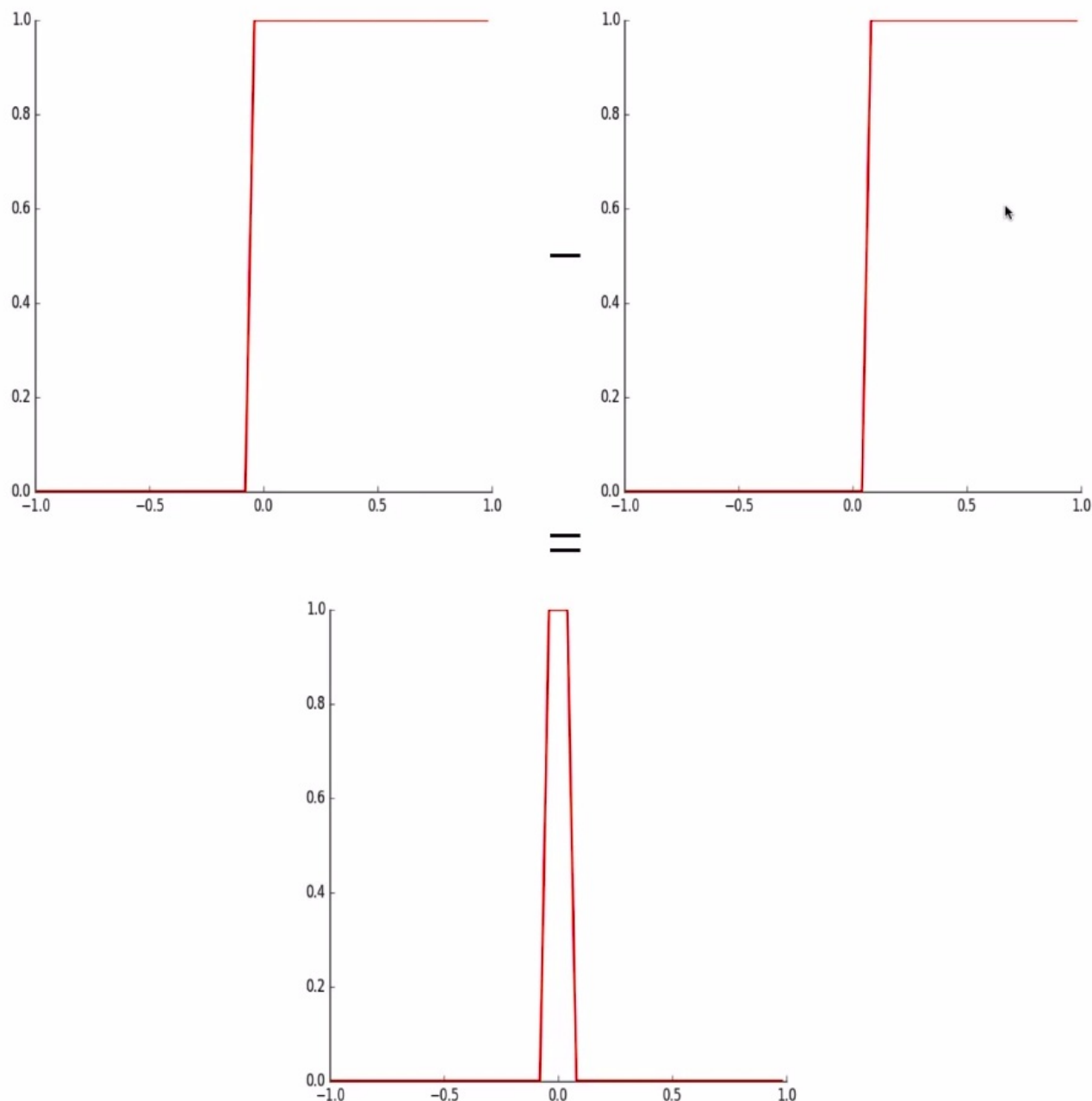
Like that combination many individual functions we will come closer to the true approximation, that's how we get the complex function we required.

when we take sigmoid function with high w and b values so the the function will be similar to step function we will take two sigmoid function like the below and subtract them.



Taking two sigmoid functions and subtracting them (Source: **onefourthlabs**).

Subtracting them leads to generation of a tower with minute width and descent height.



So subtracting then will generate a tower (Source: **onefourthlabs**).

Like this combination many towers lead to a network of neurons finally, like this we lead to an approximation of the true function i.e., the complex function finally.

This is illustrative proof for the **Universal Approximation Theorem**.

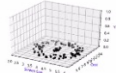
Finally we come with a conclusion as below

Take-aways



Real inputs

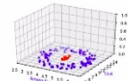
$$\in \mathbb{R}$$



$$loss = \sum_i (y_i - \hat{y}_i)^2$$



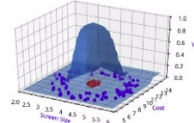
Classification



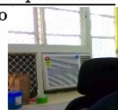
$$w = w + \eta \frac{\partial L}{\partial w}$$

$$b = b + \eta \frac{\partial L}{\partial b}$$

Let's Do Fourth Labs



$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$



Conclusion (Source: **onefourthlabs**).

This is about the power of function use to classify non linearly separable data using complex functions with help of building blocks called sigmoid neurons.

This is a small try ,uploading the notes . I believe in **"Sharing knowledge is that best way of developing skills"**.Comments will be appreciated. Even small edits can be suggested.

Each Applause will be a great encouragement.

Do follow my medium for more updates.....